Previous: [Terminal Input](file:///Users/zoltickb/Downloads2/Terminal-Input.html#Terminal-Input), Up: [Basic Input and Output](file:///Users/zoltickb/Downloads2/Basic-Input-and-Output.html#Basic-Input-and-Output)   [[Contents](file:///Users/zoltickb/Downloads2/index.html#SEC_Contents)][[Index](file:///Users/zoltickb/Downloads2/Concept-Index.html#Concept-Index)]

#### 14.1.3 Simple File I/O

The save and load commands allow data to be written to and read from disk files in various formats. The default format of files written by the save command can be controlled using the functions save\_default\_options and save\_precision.

As an example the following code creates a 3-by-3 matrix and saves it to the file ‘myfile.mat’.

A = [ 1:3; 4:6; 7:9 ];

save myfile.mat A

Once one or more variables have been saved to a file, they can be read into memory using the load command.

load myfile.mat

A

-| A =

-|

-| 1 2 3

-| 4 5 6

-| 7 8 9

: **save** file

: **save** options file

: **save** options file v1 v2 …

: **save** options file -struct STRUCT f1 f2 …

: **save** *"-"* v1 v2 …

: s = **save** (*"-"* v1 v2 …)

Save the named variables v1, v2, …, in the file file.

The special filename ‘-’ may be used to return the content of the variables as a string. If no variable names are listed, Octave saves all the variables in the current scope. Otherwise, full variable names or pattern syntax can be used to specify the variables to save. If the -struct modifier is used, fields f1 f2 … of the scalar structure STRUCT are saved as if they were variables with corresponding names. Valid options for the save command are listed in the following table. Options that modify the output format override the format specified by save\_default\_options.

If save is invoked using the functional form

save ("-option1", …, "file", "v1", …)

then the options, file, and variable name arguments (v1, …) must be specified as character strings.

If called with a filename of "-", write the output to stdout if nargout is 0, otherwise return the output in a character string.

-append

Append to the destination instead of overwriting.

-ascii

Save a single matrix in a text file without header or any other information.

-binary

Save the data in Octave’s binary data format.

-float-binary

Save the data in Octave’s binary data format but only using single precision. Only use this format if you know that all the values to be saved can be represented in single precision.

-hdf5

Save the data in HDF5 format. (HDF5 is a free, portable binary format developed by the National Center for Supercomputing Applications at the University of Illinois.) This format is only available if Octave was built with a link to the HDF5 libraries.

-float-hdf5

Save the data in HDF5 format but only using single precision. Only use this format if you know that all the values to be saved can be represented in single precision.

-V7

-v7

-7

-mat7-binary

Save the data in MATLAB’s v7 binary data format.

-V6

-v6

-6

-mat

-mat-binary

Save the data in MATLAB’s v6 binary data format.

-V4

-v4

-4

-mat4-binary

Save the data in the binary format written by MATLAB version 4.

-text

Save the data in Octave’s text data format. (default).

-zip

-z

Use the gzip algorithm to compress the file. This works equally on files that are compressed with gzip outside of octave, and gzip can equally be used to convert the files for backward compatibility. This option is only available if Octave was built with a link to the zlib libraries.

The list of variables to save may use wildcard patterns containing the following special characters:

?

Match any single character.

\*

Match zero or more characters.

[ list ]

Match the list of characters specified by list. If the first character is ! or ^, match all characters except those specified by list. For example, the pattern [a-zA-Z] will match all lower and uppercase alphabetic characters.

Wildcards may also be used in the field name specifications when using the -struct modifier (but not in the struct name itself).

Except when using the MATLAB binary data file format or the ‘-ascii’ format, saving global variables also saves the global status of the variable. If the variable is restored at a later time using ‘load’, it will be restored as a global variable.

The command

save -binary data a b\*

saves the variable ‘a’ and all variables beginning with ‘b’ to the file data in Octave’s binary format.

**See also:** [load](#XREFload), [save\_default\_options](#XREFsave_005fdefault_005foptions), [save\_header\_format\_string](#XREFsave_005fheader_005fformat_005fstri), [dlmread](#XREFdlmread), [csvread](#XREFcsvread), [fread](file:///Users/zoltickb/Downloads2/Binary-I_002fO.html#XREFfread).

There are three functions that modify the behavior of save.

: val = **save\_default\_options** ()

: old\_val = **save\_default\_options** (new\_val)

: **save\_default\_options** (new\_val, "local")

Query or set the internal variable that specifies the default options for the save command, and defines the default format.

Typical values include "-ascii", "-text -zip". The default value is -text.

When called from inside a function with the "local" option, the variable is changed locally for the function and any subroutines it calls. The original variable value is restored when exiting the function.

**See also:** [save](#XREFsave).

: val = **save\_precision** ()

: old\_val = **save\_precision** (new\_val)

: **save\_precision** (new\_val, "local")

Query or set the internal variable that specifies the number of digits to keep when saving data in text format.

When called from inside a function with the "local" option, the variable is changed locally for the function and any subroutines it calls. The original variable value is restored when exiting the function.

: val = **save\_header\_format\_string** ()

: old\_val = **save\_header\_format\_string** (new\_val)

: **save\_header\_format\_string** (new\_val, "local")

Query or set the internal variable that specifies the format string used for the comment line written at the beginning of text-format data files saved by Octave.

The format string is passed to strftime and should begin with the character ‘#’ and contain no newline characters. If the value of save\_header\_format\_string is the empty string, the header comment is omitted from text-format data files. The default value is

"# Created by Octave VERSION, %a %b %d %H:%M:%S %Y %Z <USER@HOST>"

When called from inside a function with the "local" option, the variable is changed locally for the function and any subroutines it calls. The original variable value is restored when exiting the function.

**See also:** [strftime](file:///Users/zoltickb/Downloads2/Timing-Utilities.html#XREFstrftime), [save](#XREFsave).

: **load** file

: **load** options file

: **load** options file v1 v2 …

: S = **load** ("options", "file", "v1", "v2", …)

: **load** file options

: **load** file options v1 v2 …

: S = **load** ("file", "options", "v1", "v2", …)

Load the named variables v1, v2, …, from the file file.

If no variables are specified then all variables found in the file will be loaded. As with save, the list of variables to extract can be full names or use a pattern syntax. The format of the file is automatically detected but may be overridden by supplying the appropriate option.

If load is invoked using the functional form

load ("-option1", …, "file", "v1", …)

then the options, file, and variable name arguments (v1, …) must be specified as character strings.

If a variable that is not marked as global is loaded from a file when a global symbol with the same name already exists, it is loaded in the global symbol table. Also, if a variable is marked as global in a file and a local symbol exists, the local symbol is moved to the global symbol table and given the value from the file.

If invoked with a single output argument, Octave returns data instead of inserting variables in the symbol table. If the data file contains only numbers (TAB- or space-delimited columns), a matrix of values is returned. Otherwise, load returns a structure with members corresponding to the names of the variables in the file.

The load command can read data stored in Octave’s text and binary formats, and MATLAB’s binary format. If compiled with zlib support, it can also load gzip-compressed files. It will automatically detect the type of file and do conversion from different floating point formats (currently only IEEE big and little endian, though other formats may be added in the future).

Valid options for load are listed in the following table.

-force

This option is accepted for backward compatibility but is ignored. Octave now overwrites variables currently in memory with those of the same name found in the file.

-ascii

Force Octave to assume the file contains columns of numbers in text format without any header or other information. Data in the file will be loaded as a single numeric matrix with the name of the variable derived from the name of the file.

-binary

Force Octave to assume the file is in Octave’s binary format.

-hdf5

Force Octave to assume the file is in HDF5 format. (HDF5 is a free, portable binary format developed by the National Center for Supercomputing Applications at the University of Illinois.) Note that Octave can read HDF5 files not created by itself, but may skip some datasets in formats that it cannot support. This format is only available if Octave was built with a link to the HDF5 libraries.

-import

This option is accepted for backward compatibility but is ignored. Octave can now support multi-dimensional HDF data and automatically modifies variable names if they are invalid Octave identifiers.

-mat

-mat-binary

-6

-v6

-7

-v7

Force Octave to assume the file is in MATLAB’s version 6 or 7 binary format.

-mat4-binary

-4

-v4

-V4

Force Octave to assume the file is in the binary format written by MATLAB version 4.

-text

Force Octave to assume the file is in Octave’s text format.

**See also:** [save](#XREFsave), [dlmwrite](#XREFdlmwrite), [csvwrite](#XREFcsvwrite), [fwrite](file:///Users/zoltickb/Downloads2/Binary-I_002fO.html#XREFfwrite).

: str = **fileread** (filename)

Read the contents of filename and return it as a string.

**See also:** [fread](file:///Users/zoltickb/Downloads2/Binary-I_002fO.html#XREFfread), [textread](#XREFtextread), [sscanf](file:///Users/zoltickb/Downloads2/Formatted-Input.html#XREFsscanf).

: **native\_float\_format** ()

Return the native floating point format as a string.

It is possible to write data to a file in a similar way to the disp function for writing data to the screen. The fdisp works just like disp except its first argument is a file pointer as created by fopen. As an example, the following code writes to data ‘myfile.txt’.

fid = fopen ("myfile.txt", "w");

fdisp (fid, "3/8 is ");

fdisp (fid, 3/8);

fclose (fid);

See [Opening and Closing Files](file:///Users/zoltickb/Downloads2/Opening-and-Closing-Files.html#Opening-and-Closing-Files), for details on how to use fopen and fclose.

: **fdisp** (fid, x)

Display the value of x on the stream fid.

For example:

fdisp (stdout, "The value of pi is:"), fdisp (stdout, pi)

-| the value of pi is:

-| 3.1416

Note that the output from fdisp always ends with a newline.

**See also:** [disp](file:///Users/zoltickb/Downloads2/Terminal-Output.html#XREFdisp).

Octave can also read and write matrices text files such as comma separated lists.

: **dlmwrite** (file, M)

: **dlmwrite** (file, M, delim, r, c)

: **dlmwrite** (file, M, key, val …)

: **dlmwrite** (file, M, "-append", …)

: **dlmwrite** (fid, …)

Write the numeric matrix M to the text file file using a delimiter.

file should be a filename or a writable file ID given by fopen.

The parameter delim specifies the delimiter to use to separate values on a row. If no delimiter is specified the comma character ‘,’ is used.

The value of r specifies the number of delimiter-only lines to add to the start of the file.

The value of c specifies the number of delimiters to prepend to each line of data.

If the argument "-append" is given, append to the end of file.

In addition, the following keyword value pairs may appear at the end of the argument list:

"append"

Either "on" or "off". See "-append" above.

"delimiter"

See delim above.

"newline"

The character(s) to separate each row. Three special cases exist for this option. "unix" is changed into "\n", "pc" is changed into "\r\n", and "mac" is changed into "\r". Any other value is used directly as the newline separator.

"roffset"

See r above.

"coffset"

See c above.

"precision"

The precision to use when writing the file. It can either be a format string (as used by fprintf) or a number of significant digits.

dlmwrite ("file.csv", reshape (1:16, 4, 4));

dlmwrite ("file.tex", a, "delimiter", "&", "newline", "\n")

**See also:** [dlmread](#XREFdlmread), [csvread](#XREFcsvread), [csvwrite](#XREFcsvwrite).

: data = **dlmread** (file)

: data = **dlmread** (file, sep)

: data = **dlmread** (file, sep, r0, c0)

: data = **dlmread** (file, sep, range)

: data = **dlmread** (…, "emptyvalue", EMPTYVAL)

Read numeric data from the text file file which uses the delimiter sep between data values.

If sep is not defined the separator between fields is determined from the file itself.

The optional scalar arguments r0 and c0 define the starting row and column of the data to be read. These values are indexed from zero, i.e., the first data row corresponds to an index of zero.

The range parameter specifies exactly which data elements are read. The first form of the parameter is a 4-element vector containing the upper left and lower right corners [R0,C0,R1,C1] where the indices are zero-based. Alternatively, a spreadsheet style form such as "A2..Q15" or "T1:AA5" can be used. The lowest alphabetical index 'A' refers to the first column. The lowest row index is 1.

file should be a filename or a file id given by fopen. In the latter case, the file is read until end of file is reached.

The "emptyvalue" option may be used to specify the value used to fill empty fields. The default is zero. Note that any non-numeric values, such as text, are also replaced by the "emptyvalue".

**See also:** [csvread](#XREFcsvread), [textscan](#XREFtextscan), [textread](#XREFtextread), [dlmwrite](#XREFdlmwrite).

: **csvwrite** (filename, x)

: **csvwrite** (filename, x, dlm\_opt1, …)

Write the numeric matrix x to the file filename in comma-separated-value (CSV) format.

This function is equivalent to

dlmwrite (filename, x, ",", dlm\_opt1, …)

Any optional arguments are passed directly to dlmwrite (see [dlmwrite](#XREFdlmwrite)).

**See also:** [csvread](#XREFcsvread), [dlmwrite](#XREFdlmwrite), [dlmread](#XREFdlmread).

: x = **csvread** (filename)

: x = **csvread** (filename, dlm\_opt1, …)

Read the comma-separated-value (CSV) file filename into the matrix x.

Note: only CSV files containing numeric data can be read.

This function is equivalent to

x = dlmread (filename, "," , dlm\_opt1, …)

Any optional arguments are passed directly to dlmread (see [dlmread](#XREFdlmread)).

**See also:** [dlmread](#XREFdlmread), [textread](#XREFtextread), [textscan](#XREFtextscan), [csvwrite](#XREFcsvwrite), [dlmwrite](#XREFdlmwrite).

Formatted data from can be read from, or written to, text files as well.

: [a, …] = **textread** (filename)

: [a, …] = **textread** (filename, format)

: [a, …] = **textread** (filename, format, n)

: [a, …] = **textread** (filename, format, prop1, value1, …)

: [a, …] = **textread** (filename, format, n, prop1, value1, …)

Read data from a text file.

The file filename is read and parsed according to format. The function behaves like strread except it works by parsing a file instead of a string. See the documentation of strread for details.

In addition to the options supported by strread, this function supports two more:

* "headerlines": The first value number of lines of filename are skipped.
* "endofline": Specify a single character or "\r\n". If no value is given, it will be inferred from the file. If set to "" (empty string) EOLs are ignored as delimiters.

The optional input n (format repeat count) specifies the number of times the format string is to be used or the number of lines to be read, whichever happens first while reading. The former is equivalent to requesting that the data output vectors should be of length N. Note that when reading files with format strings referring to multiple lines, n should rather be the number of lines to be read than the number of format string uses.

If the format string is empty (not just omitted) and the file contains only numeric data (excluding headerlines), textread will return a rectangular matrix with the number of columns matching the number of numeric fields on the first data line of the file. Empty fields are returned as zero values.

Examples:

Assume a data file like:

1 a 2 b

3 c 4 d

5 e

[a, b] = textread (f, "%f %s")

returns two columns of data, one with doubles, the other a

cellstr array:

a = [1; 2; 3; 4; 5]

b = {"a"; "b"; "c"; "d"; "e"}

[a, b] = textread (f, "%f %s", 3)

(read data into two culumns, try to use the format string

three times)

returns

a = [1; 2; 3]

b = {"a"; "b"; "c"}

With a data file like:

1

a

2

b

[a, b] = textread (f, "%f %s", 2)

returns a = 1 and b = {"a"}; i.e., the format string is used

only once because the format string refers to 2 lines of the

data file. To obtain 2x1 data output columns, specify N = 4

(number of data lines containing all requested data) rather

than 2.

**See also:** [strread](file:///Users/zoltickb/Downloads2/Manipulating-Strings.html#XREFstrread), [load](#XREFload), [dlmread](#XREFdlmread), [fscanf](file:///Users/zoltickb/Downloads2/Formatted-Input.html#XREFfscanf), [textscan](#XREFtextscan).

: C = **textscan** (fid, format)

: C = **textscan** (fid, format, repeat)

: C = **textscan** (fid, format, param, value, …)

: C = **textscan** (fid, format, repeat, param, value, …)

: C = **textscan** (str, …)

: [C, position, errmsg] = **textscan** (…)

Read data from a text file or string.

The string str or file associated with fid is read from and parsed according to format. The function is an extension of strread and textread. Differences include: the ability to read from either a file or a string, additional options, and additional format specifiers.

The input is interpreted as a sequence of words, delimiters (such as whitespace), and literals. The characters that form delimiters and whitespace are determined by the options. The format consists of format specifiers interspersed between literals. In the format, whitespace forms a delimiter between consecutive literals, but is otherwise ignored.

The output C is a cell array where the number of columns is determined by the number of format specifiers.

The first word of the input is matched to the first specifier of the format and placed in the first column of the output; the second is matched to the second specifier and placed in the second column and so forth. If there are more words than specifiers then the process is repeated until all words have been processed or the limit imposed by repeat has been met (see below).

The string format describes how the words in str should be parsed. As in fscanf, any (non-whitespace) text in the format that is not one of these specifiers is considered a literal. If there is a literal between two format specifiers then that same literal must appear in the input stream between the matching words.

The following specifiers are valid:

%f

%f64

%n

The word is parsed as a number and converted to double.

%f32

The word is parsed as a number and converted to single (float).

%d

%d8

%d16

%d32

%d64

The word is parsed as a number and converted to int8, int16, int32, or int64. If no size is specified then int32 is used.

%u

%u8

%u16

%u32

%u64

The word is parsed as a number and converted to uint8, uint16, uint32, or uint64. If no size is specified then uint32 is used.

%s

The word is parsed as a string ending at the last character before whitespace, an end-of-line, or a delimiter specified in the options.

%q

The word is parsed as a "quoted string". If the first character of the string is a double quote (") then the string includes everything until a matching double quote—including whitespace, delimiters, and end-of-line characters. If a pair of consecutive double quotes appears in the input, it is replaced in the output by a single double quote. For examples, the input "He said ""Hello""" would return the value ’He said "Hello"’.

%c

The next character of the input is read. This includes delimiters, whitespace, and end-of-line characters.

%[…]

%[^…]

In the first form, the word consists of the longest run consisting of only characters between the brackets. Ranges of characters can be specified by a hyphen; for example, %[0-9a-zA-Z] matches all alphanumeric characters (if the underlying character set is ASCII). Since MATLAB treats hyphens literally, this expansion only applies to alphanumeric characters. To include ’-’ in the set, it should appear first or last in the brackets; to include ’]’, it should be the first character. If the first character is ’^’ then the word consists of characters **not** listed.

%N…

For %s, %c %d, %f, %n, %u, an optional width can be specified as %Ns, etc. where N is an integer > 1. For %c, this causes exactly N characters to be read instead of a single character. For the other specifiers, it is an upper bound on the number of characters read; normal delimiters can cause fewer characters to be read. For complex numbers, this limit applies to the real and imaginary components individually. For %f and %n, format specifiers like %N.Mf are allowed, where M is an upper bound on number of characters after the decimal point to be considered; subsequent digits are skipped. For example, the specifier %8.2f would read 12.345e6 as 1.234e7.

%\*…

The word specified by the remainder of the conversion specifier is skipped.

literals

In addition the format may contain literal character strings; these will be skipped during reading. If the input string does not match this literal, the processing terminates.

Parsed words corresponding to the first specifier are returned in the first output argument and likewise for the rest of the specifiers.

By default, if there is only one input argument, format is "%f". This means that numbers are read from the input into a single column vector. If format is explicitly empty ("") then textscan will return data in a number of columns matching the number of fields on the first data line of the input. Either of these is suitable only when the input is exclusively numeric.

For example, the string

str = "\

Bunny Bugs 5.5\n\

Duck Daffy -7.5e-5\n\

Penguin Tux 6"

can be read using

a = textscan (str, "%s %s %f");

The optional numeric argument repeat can be used for limiting the number of items read:

-1

Read all of the string or file until the end (default).

N

Read until the first of two conditions occurs: 1) the format has been processed N times, or 2) N lines of the input have been processed. Zero (0) is an acceptable value for repeat. Currently, end-of-line characters inside %q, %c, and %[…]$ conversions do not contribute to the line count. This is incompatible with MATLAB and may change in future.

The behavior of textscan can be changed via property/value pairs. The following properties are recognized:

"BufSize"

This specifies the number of bytes to use for the internal buffer. A modest speed improvement may be obtained by setting this to a large value when reading a large file, especially if the input contains long strings. The default is 4096, or a value dependent on n if that is specified.

"CollectOutput"

A value of 1 or true instructs textscan to concatenate consecutive columns of the same class in the output cell array. A value of 0 or false (default) leaves output in distinct columns.

"CommentStyle"

Specify parts of the input which are considered comments and will be skipped. value is the comment style and can be either (1) A string or 1x1 cell string, to skip everything to the right of it; (2) A cell array of two strings, to skip everything between the first and second strings. Comments are only parsed where whitespace is accepted and do not act as delimiters.

"Delimiter"

If value is a string, any character in value will be used to split the input into words. If value is a cell array of strings, any string in the array will be used to split the input into words. (default value = any whitespace.)

"EmptyValue"

Value to return for empty numeric values in non-whitespace delimited data. The default is NaN. When the data type does not support NaN (int32 for example), then the default is zero.

"EndOfLine"

value can be either an emtpy or one character specifying the end-of-line character, or the pair "\r\n" (CRLF). In the latter case, any of "\r", "\n" or "\r\n" is counted as a (single) newline. If no value is given, "\r\n" is used.

"HeaderLines"

The first value number of lines of fid are skipped. Note that this does not refer to the first non-comment lines, but the first lines of any type.

"MultipleDelimsAsOne"

If value is nonzero, treat a series of consecutive delimiters, without whitespace in between, as a single delimiter. Consecutive delimiter series need not be vertically aligned. Without this option, a single delimiter before the end of the line does not cause the line to be considered to end with an empty value, but a single delimiter at the start of a line causes the line to be considered to start with an empty value.

"TreatAsEmpty"

Treat single occurrences (surrounded by delimiters or whitespace) of the string(s) in value as missing values.

"ReturnOnError"

If set to numerical 1 or true, return normally as soon as an error is encountered, such as trying to read a string using %f. If set to 0 or false, return an error and no data.

"Whitespace"

Any character in value will be interpreted as whitespace and trimmed; The default value for whitespace is " \b\r\n\t" (note the space). Unless whitespace is set to "" (empty) AND at least one "%s" format conversion specifier is supplied, a space is always part of whitespace.

When the number of words in str or fid doesn’t match an exact multiple of the number of format conversion specifiers, textscan’s behavior depends on whether the last character of the string or file is an end-of-line as specified by the EndOfLine option:

last character = end-of-line

Data columns are padded with empty fields, NaN or 0 (for integer fields) so that all columns have equal length

last character is not end-of-line

Data columns are not padded; textscan returns columns of unequal length

The second output position provides the location, in characters from the beginning of the file or string, where processing stopped.

**See also:** [dlmread](#XREFdlmread), [fscanf](file:///Users/zoltickb/Downloads2/Formatted-Input.html#XREFfscanf), [load](#XREFload), [strread](file:///Users/zoltickb/Downloads2/Manipulating-Strings.html#XREFstrread), [textread](#XREFtextread).

The importdata function has the ability to work with a wide variety of data.

: A = **importdata** (fname)

: A = **importdata** (fname, delimiter)

: A = **importdata** (fname, delimiter, header\_rows)

: [A, delimiter] = **importdata** (…)

: [A, delimiter, header\_rows] = **importdata** (…)

Import data from the file fname.

Input parameters:

* fname The name of the file containing data.
* delimiter The character separating columns of data. Use \t for tab. (Only valid for ASCII files)
* header\_rows The number of header rows before the data begins. (Only valid for ASCII files)

Different file types are supported:

* ASCII table

Import ASCII table using the specified number of header rows and the specified delimiter.

* Image file
* MATLAB file
* Spreadsheet files (depending on external software)
* WAV file

**See also:** [textscan](#XREFtextscan), [dlmread](#XREFdlmread), [csvread](#XREFcsvread), [load](#XREFload).

|  |  |  |
| --- | --- | --- |
| • [Saving Data on Unexpected Exits](file:///Users/zoltickb/Downloads2/Saving-Data-on-Unexpected-Exits.html#Saving-Data-on-Unexpected-Exits): |  |  |

Previous: [Terminal Input](file:///Users/zoltickb/Downloads2/Terminal-Input.html#Terminal-Input), Up: [Basic Input and Output](file:///Users/zoltickb/Downloads2/Basic-Input-and-Output.html#Basic-Input-and-Output)   [[Contents](file:///Users/zoltickb/Downloads2/index.html#SEC_Contents)][[Index](file:///Users/zoltickb/Downloads2/Concept-Index.html#Concept-Index)]